# Empirical Study on Overlapping Community Detection in Question and Answer Sites

Zide Meng, Fabien Gandon, Catherine Faron Zucker, Ge Song

# Empirical Study on Overlapping Community Detection in Question and Answer Sites

Zide Meng*, Fabien Gandon*, Catherine Faron Zucker† and Ge Song*
*INRIA Sophia Antipolis Méditerranée, 06900 Sophia Antipolis, France
Email: {zide.meng, fabien.gandon, ge.song}@inria.fr
†Univ. Nice Sophia Antipolis, CNRS, I3S, UMR 7271, 06900 Sophia Antipolis, France
Email: faron@unice.fr

***Abstract*—In many social networks, people interact based on their interests. Community detection algorithms are then useful to reveal the sub-structures of a network and help us find interest groups. Identifying these social communities can bring benefit to understanding and predicting users behaviors. However, for some kind of online community sites such as question-and-answer (Q&A) sites or forums, there is no friendship based social network structure, which means people are not aware who they are in contact with. Therefore, many traditional community detection techniques do not apply directly. In this paper, we propose an empirical approach for extracting data from Q&A sites suitable to apply community detection methods. Then we compare three kinds of community detection methods we applied on a dataset extracted from the popular Q&A site StackOverflow. We analyze and comment the results of each method.**

## I. Introduction

Question-and-answer (Q&A) sites initially aimed at providing platforms where users could ask for help from some professionals. Since these questions and answers can be viewed and searched afterwards, people with similar questions can also directly find solutions by browsing the site. Gradually, Q&A sites have become huge repositories which provide highly reusable and highly valuable knowledge[1].

A large number of people are very active and keep contributing answers to these sites. Most of them are more likely to answer questions about topics in which they are interested and specialized. Undoubtedly, one of the most important resources of Q&A sites are people with professional domain knowledge or so-called experts. Several proposals have been done to detect those experts [1][2].

However, rather than focusing on individual experts, we try to answer this question: how to identify these interest groups and their topics in Q&A sites? Detecting interest groups can contribute to the question routing problem [3], which is very important in Q&A site optimization. It can also contribute to the community management, for instance by allowing to monitor the interest evolution or community evolution in Q&A sites. Many community detection algorithms have been developed to find sub-structures in social networks. Q&A sites are also social networks. However, unlike friendship networks such as Facebook, there are no explicit relationships between people on Q&A sites. Besides people are not aware of who they are interacting with, and normally they do not maintain a solid relationship. So, compared with normal social network, there are more *star* structures than *triangle* structures in Q&A sites. People are more like isolated nodes grouped by interests. So interest groups are an important implicit sub-structure in such social sites. Moreover, people have multiple interests and therefore belong to several interest groups. Therefore an important aspect of this question is the ability to detect overlapping communities.

In this paper, we first introduce an empirical method to enrich question tags. We then propose a tag prefix tree model to extract topics from question tags. We then use these topics to detect interest groups. We first survey the state-of-the-art graph-based community approaches in Q&A sites, and point out the difference between Q&A social graphs and other social graphs. Our results show that our approach of interest group detection can better answer our research question.

The rest of this paper is organized as follows. In Section II we present some related works and their limitations for community detection on Q&A sites. In Section III we introduce our method for interest group detection in Q&A sites. In Section IV we describe some experiments we have conducted and we present a comparative evaluation of our method. In Section V we summarize our contribution and we point out some limitations and future works.

## II. Related Work

We distinguish between three kinds of approaches for community detection depending on their input: the question-answer or co-answer social network; the similarity of user questions or answers; a model of users, questions and tags.

A first and direct solution is to extract an implicit network structure (such as question-answer network, co-answer network, etc.) from interaction traces to come down to a traditional community detection problem on social networks. Since intuitively, users are grouped by interests, and most of their interactions are based on shared interests, it is reasonable to induce a network structure from these interactions and then run community detection algorithms on the network. Many classical algorithms have been developed such as [4], [5]. There are many constraints when adopting these methods. First, they do not take into account node attributes nor link attributes. Take co-answer networks as an example, where nodes represent users and links represent users answering the same questions. In case two users are connected, these methods can only indicate that they have answered the same

[1]Some examples of Q&A web sites:
http://en.wikipedia.org/wiki/List_of_question-and-answer_websites

TABLE I: Comparison of the main approaches and our method

| | uses nodes | uses links | overlap | membership |
|---|---|---|---|---|
| graph-based | no | yes | yes/no | yes/no |
| clustering-based | yes | no | no | no |
| model-based | yes | yes | yes | yes |
| our-method | yes | yes | yes | yes |

questions many times. They cannot provide the information whether they have answered questions on the same topic or on different topics. Second, the result is largely determined by the network structure. If the co-answer network has an 'octopus' or 'jellfish' shape [6], these methods may fail. Third, most of the works adopting this approach cannot detect overlapping communities; only recent work, such as [4] begin to address this problem.

This problem can also be envisioned as a clustering problem. By computing similarities between user profiles, we detect groups according to clustering results. The choice of the similarity metrics is quite important and largely influences clustering results. Clustering methods, such as [7] or [8], group users according to their features. They do not take the network structure into consideration. Moreover, these algorithms produce hard partitions, that is to say, a user can only belong to one interest group, which is too restrictive.

A third approach consists in using a *model* for both the user profiles and the network structure to solve community detection and link prediction. These methods normally use Bayes network to model different aspects of a social network. For example, [9] use a LDA-based method on social tagging systems where users label resources with tags, but they do not consider the problem of *overlapping* community detection. [10] use an extended LDA-based model to analyze academic social networks in order to find expert authors, papers and conferences. We adapted this approach by modeling users, topics and tags rather than modeling authors, conferences and papers. This is further explained in the next section.

Table I summarizes the main features of the three approaches. To sum up, graph-based approaches normally use link information while ignoring node attributes. Most of them cannot detect overlapping communities, but recent works do it, such as [4], and provide membership ratios which are weights denoting to what extent a user belongs to a community. Clustering-based approaches use node attributes to group similar users. Normally their results are hard-partitioned communities, with no overlapping and no membership information. Model-based approaches overcome the shortcomings of graph-based and clustering-based approaches, using both node attributes and link information.

## III. MODELS AND SOLUTIONS

### A. Problem Definition

In StackOverflow[2], a user submits a question, then assigns 1∼5 tags to indicate the key topics of this question. We compute the tag list of a question, sorted by the global occurrence of each tag. Other users who are interested in the question may provide answers to the question or comments to other answers. As tags attached to a question can reflect its domain, users answering the question can be considered as interested by this domain. Let $U = \{u_1, u_2 ... u_n\}$ be the set of users, $Q = \{q_1, q_2 ... q_m\}$ the set of questions and $T = \{t_1, t_2 ... t_v\}$ the set of tags. We aim at (1) extracting topics $Topic = \{topic_1, topic_2 ... topic_k\}$ from $T$, and for each $topic_k \in Topic$, defining $topic_k = \{p_{ki}, p_{kl} ... p_{kj}\}$ where $p_{ki}$ denotes the probability of tag $t_i$ to be related to $topic_k$; and then (2) detecting user interests. For a user $u_i \in U$, we define $I_i = \{I_{i1}, I_{i2} ... I_{ik}\}$ where $I_{ik}$ denotes the probability of $u_i$ to be related to $topic_k$. Here, we assume that if a user is related to $topic_k$, then this user can be assigned to an interest group (changeable with 'community') where the topic of this group is $topic_k$.

### B. Question Tag Enrichment

We have empirically found that the first tag of a question normally indicates the domain of the question. For example, a question tagged with '*c++ iostream fstream*' is related to *c++*; A question tagged with '*html css height*' is related to *html*. However, there are also some questions that only have less and low popular tags, like a question tagged with '*ant*' or a question tagged with '*qt boost*'. For these questions, the domain is not obvious. To answer this problem, we propose a tag enrichment method implemented in Algorithm 1.

**Algorithm 1: Tag Enrichment Algorithm**

```
input: tag list of questions
output: enriched tag list of questions
/*pre-process*/
tag_domaintags_map={}
foreach question taglist:
    first_tag=taglist[0]
    foreach tag in taglist:
        if not tag_domaintags_map.contain(tag):
            tag_domaintags_map[tag]={}
        if tag_domaintags_map[tag].contain(first_tag):
            tag_domaintags_map[tag][first_tag]++
        else
            tag_domaintags_map[tag][first_tag]=1
foreach tag,domaintags in tag_domaintags_map:
    foreach first_tag,freq in domaintags:
    normalize=freq/sum(freq) **
        tag_domaintags_map[tag][first_tag]=normalize
/*post-process*/
foreach question's taglist:
    temp_first_tag_map={}
    foreach tag in taglist:
        discount=1
        get top 5 first_tag from tag_domaintags_map[tag]
        foreach first_tag,value in top 5:
            value=value*discount, discount*=0.5
            if temp_first_tag_map.contain(first_tag):
                temp_first_tag_map[first_tag]+ =value
            else
                temp_first_tag_map[first_tag]=value
    enrich_tag=get top first tag from temp_first_tag_map
    if enrich_tag != taglist[0]:
        taglist.insert(enrich_tag)
```

** In order to lower the probability of low frequency tag as first tag, we actually use normalize=freq/*sum*(freq)*sigmoid(first_tag's frequency)

[2]http://www.stackoverflow.com/

In pre-process, for each tag in a question, we compute the frequency of the first tags of the question tag lists where it occurs and we record it in a hashmap. For example, with the three question tag lists '*html css height*', '*html css layout*', and '*c# gui layout*', we record for tag *html* the frequency map {*html*:2}, for tag *css* the frequency map {*html*:2}, and for tag *layout* the frequency map {*html*:1,*c#*:1}. After processing with all question tags, we normalize the frequencies: the frequency map of tag *css* becomes {*html*:1.0} and the frequency map of *layout* becomes {*html*:0.5,*c#*:0.5}. In order to lower the probability of low frequency tag as first tag, we use the equation 1:

$$normalize(tag) = \frac{record\_freq}{sum(record\_freq)} * sigmoid(frequency) \tag{1}$$

Here, *record_freq* denotes the recorded frequency of the tag, *sum(record_freq)* denotes the sum of these recorded frequencies, *frequency* denotes the total frequency of the tag. For example, let us consider the frequency map {*html*:10, *jquery*:2}; for normalizing the frequency of tag *html* in this map, *record_freq* is 10, *sum(record_freq)* is 12, *frequency* is 5552 (which is the number of occurrences of tag *html* in all the questions). The sigmoid function is $\frac{1}{(1+e^{-k*z})}$, where $k$ is chosen equal to 0.001. This function is used as a squashing function for numerical stability. As a result, for each tag, we return a list of enriching tags with their probabilities.

In post-processing, given a question tag list, we fetch the top 5 enriching tags for each tag. then we accumulate the corresponding probabilities with a discount taking the tag position into account. Then we consider the tag with the highest probability as the enriching tag. If this tag already exists in the original tag list, we simply skip the insertion, or else we insert it at the first position. We processed 242552 question tag lists, and our algorithm enriched 33622 of them (13.5%). Table II presents the results of the enrichment of some tag lists (added tags are highlighted in bold).

TABLE II: Original and enriched tag lists

| original tag list | enriched tag list |
|---|---|
| ant | **java**, ant |
| qt, boost | **c++**, qt, boost |
| django, hosting | **python**, django, hosting |
| xslt, dynamic, xsl | **xml**, xslt, dynamic, xsl |
| sql-server-2005, sorting | **sql**, sql-server-2005, sorting |
| tomcat, grails, connection | **java**, tomcat, grails, connection |
| cocoa, osx, mac, plugins | **objective-c**, cocoa, osx, mac, plugins |
| spring, j2ee, module, count | **java**, spring, j2ee, module, count |

## C. Topic Extraction

From the observation of our dataset, we empirically stated that high frequency tags are more generic and low frequency tags are more specific, and most of the low frequency tags are related to a more generic tag. For example, for a question tagged with '*c++ iostream fstream*' (with tags sorted according their frequency), we could find that it was related to *c++* and to the *iostream* topic of *c++*, and more specifically, it focused on *fstream*. This inspired us to build a tag prefix tree to extract topics and compute the probability for a tag to be related to a topic. We describe the process in Algorithm 2.

In the *build trees* process, we build a tag prefix tree according to the position of tags in a question, and record the occurrence of each node. For example, let us consider three question tag lists: '*html css height*', '*html css layout*', and '*c# gui layout*'. We build two trees. The root of the first tree is '*html*', the occurrence of this node is *2*, it has only one child '*css*', which occurrence is *2*, and this node has itself two children, '*layout*' and '*height*', the occurrence of each of them is *1*. The root of the second tree is '*c#*' with *1* occurrence. By processing all question tag lists, many trees are generated with different sizes. We construct an affinity matrix only for root nodes. The similarity of two nodes is computed as $\#(root\_i, root\_j)/(\#root\_i + \#root\_j)$, where $\#(root\_i, root\_j)$ denotes the co-occurrence of tags $\#root\_i$ and $\#root\_j$, and $\#root\_i$ and $\#root\_j$ denote the occurrence of tag $\#root\_i$ and tag $\#root\_j$ separately. Then we run spectral clustering [11] on the affinity matrix to group these root nodes into topics. As it requires the number of topics, we choose the same number *30* as [12], which has proved to be a reasonable setting for the Stackoverflow dataset. We then combine trees if their root nodes belong to the same topic. It leads to a result where each tree represents a topic. Therefore, in the *compute topic-tag distribution* process, for each topic tree, we recursively compute each tag probability to belong to this topic. We compute the probability of the $i^{th}$ child tag $p\_sub\_i$ of a tag as described in Equation 2, where $p$ and $\#p$ denote the probability and the occurrence of the parent tag, and $\#sub\_i$ denote the occurrence of the $i^{th}$ child tag.

$$p\_sub\_i = \frac{\#sub_i}{\#p} * p \tag{2}$$

**Algorithm 2: Topic Extraction Algorithm**

```
input: enriched tag list of questions
output: topic-tag distribution
/*build trees process*/
trees=[0,{}] /* tag frequency and subtree */
foreach question's taglist:
    cur_tree=trees
    foreach tag in taglist:
        if cur_tree[1].contain(tag):
            cur_tree[1][tag][0]+=1
        else
            cur_tree[1][each]=[1,{}]
        cur_tree=cur_tree[1][each]
/*build affinity matrix for root_tags*/
root_tags=trees[1].keyset()
root_tags_affinities=[#root_tags][#root_tags]
foreach root_i in root_tags:
    foreach root_j in root_tags:
        value=#(root_i,root_j)/(#root_i+#root_i)
        root_tags_affinities[root_i][root_j]=value
groups=clustering(root_tags_affinities) **
/*combine tree process*/
newtrees=[0,{}]
foreach group_root_taglist in groups
    subtrees=[0,{}]
    foreach root_tag in group_root_taglist:
        subtrees[0]+=trees[1][root_tag][0]
        subtrees[1][root_tag]=trees[1][root_tag]
    newtrees[1][groupid]=subtrees
/*compute topic-tag distribution*/
all_distributions=[]
foreach groupid in newtrees[1].keyset()
    distribtuion={}
    total=trees[1][groupid][0]
    compute(groupid,total,trees[1][tag],distribtuion)
    all_distributions.append(distribtuion)
/*sub function to compute topic-tag distribution*/
define compute(tag,total,tree,distrib):
    if tree[1]!=null:
        foreach tag,sub_tree in tree[1].items():
            compute(tag,total,sub_tree,distrib)
    if distrib.contain(tag):
        distrib[tag]+=tree[0]/float(total)
    else
    distrib[tag]=tree[0]/float(total)
```

** we use spectral clustering to divide these root tags into several groups

Table III compares some results of this process with the state-of-the-art topic model LDA. We used the same topic number *30* for LDA. We can find that both results are very relevant to topics. However, our model does not require many iterations to converge, which makes it more efficient. We also need to point out that we use the spectral clustering algorithm in a step of our method. We used the implementation of this algorithm from scikit-learn toolkit[3]. But we only run it on the root node, which has quite a small size, so the extra cost is acceptable.

TABLE III: Top tags and their probabilities for topics *iphone*, *sql* and *linux* computed by our method and LDA method

| our method | LDA |
|---|---|
| (iphone, 0.300), (objective-c, 0.147), (iphone-sdk, 0.088), (cocoa-touch, 0.087), (cocoa, 0.073), (xcode, 0.029), (uikit, 0.012), (uitableview, 0.011), (osx, 0.010) | (cocoa, 0.182), (objective-c, 0.173), (iphone, 0.0795), (cocoa-touch, 0.048), (iphone-sdk, 0.034), (mac, 0.028), (osx, 0.027), (xcode, 0.018), (memory-management, 0.013) |
| (sql, 0.185), (sql-server, 0.157), (mysql, 0.078), (database, 0.069), (sql-server-2005, 0.046), (tsql, 0.032), (oracle, 0.018), (query, 0.017), (stored-procedures, 0.015) | (sql-server, 0.216), (sql, 0.198), (sql-server-2005, 0.061), (tsql, 0.055), (database, 0.052), (stored-procedures, 0.024), (database-design, 0.020), (performance, 0.016), (c#, 0.016) |
| (linux, 0.292), (bash, 0.088), (unix, 0.070), (shell, 0.048), (scripting, 0.023), (command-line, 0.019), (ubuntu, 0.016), (belongs-on-serverfault, 0.013), (shell-script, 0.012) | (linux, 0.074), (c, 0.058), (bash, 0.049), (unix, 0.042), (perl, 0.032), (shell, 0.030), (vim, 0.027), (regex, 0.024), (c++, 0.016) |

### D. Overlapping Interest Group Detection

With StackOverflow data, a starting point for community detection is to model the initial situation as follows: a user answering a question acquires the tags attached to this question and gradually, each user acquires a list of tags. So we use a tag list to represent a user: $U = \{U_i | i = 1, ..., n\}, U_i = \{tag_i | i = m, n, ..., k\}$. Then our goal is for $U_i$, to find $I_i = \{I_{i1}, I_{i2}...I_{ik}\}$ where $I_{ik}$ denotes the probability of user $U_i$ to be related to $topic_k$. As we already have topic-tag distribution (see section III-C), we simply compute user interests with Equation 3 where $P_{t,k}$ denotes the probability of tag $t$ to be related to topic $k$. Then a user will be assigned to an interest group with $topic_k$, if $P_{t,k}$ is greater than a threshold.

$$I_{i,k} = \sum_{t=1}^{v} \begin{cases} P_{t,k} & \text{if } tag_t \in U_i \\ 0 & \text{if } tag_t \notin U_i \end{cases} \tag{3}$$

## IV. Experiments and Evaluation on StackOverflow data

We conducted experiments on a dataset from the Q&A site StackOverflow to evaluate the performance of our approach compared to three other community detection algorithms.

### A. Dataset and Protocol

According to the dataset, the total number of users is 103K. Among them, 47K users submitted at least one question, and 54K users answered at least one question. The total number of tags attached to questions is 24K, and 20% of them are used more than 10 times. The total number of posts is 1.1M, among which they are 242K questioners and 870K answers.

Traditional community detection algorithms are based on network structure. We extracted a co-answer network inspired by the notion of co-view network introduced in [8]'s work. The idea behind it is that if two users answer the same questions they have at least one common interest on this question. Therefore, they share some of their interests. So, this co-answer network, to some extent, can reflect the co-interests of users. Then we filtered the co-answer links with a rule stating that a link is kept if two users answer the same questions more than 10, 15, 20 or 25 times. As a result, we obtained four noise-less datasets. We run the experiments on a computer with 3GHz Intel i7 CPU and 8GB RAM.

### B. Evaluation of Our Method

We run our approach on the co_answer_10 dataset, For the topic number $K$, we chose the same setting *30* as in [12]'s work. Table IV shows some users and their top 10 tags. The first row contains user ids, the second row contains their detected interest groups with their probability. The following ten rows show the top 10 tags for each user. We replaced group ids with names assigned according to tags in each group.

TABLE IV: Proposed empirical model results

| user_10224 | user_103043 | user_113570 |
|---|---|---|
| database(0.805), c#-dev(0.081) | java-dev(0.664), database(0.105) | c#-dev(0.393), web-dev(0.328) |
| sql-server(21) | java(135) | c#(107) |
| sql(21) | swing(28) | jquery(89) |
| tsql(6) | oracle(27) | javascript(56) |
| performance(4) | sql(23) | .net(47) |
| database(4) | subjective(15) | asp.net(27) |
| stored-procedures(3) | windows(13) | css(23) |
| sql-server-2005(3) | eclipse(12) | regex(20) |
| .net(3) | best-practices(12) | html(20) |
| mysql(2) | plsql(10) | iphone(12) |
| sql-server-2000(2) | regex(10) | string(10) |
| **user_24181** | **user_34509** | **user_30461** |
| web-dev(0.743), database(0.072) | c-dev(0.663), linux-dev(0.083) | ios-dev(0.885), linux-dev(0.020) |
| php(304) | c++(703) | cocoa(333) |
| javascript(193) | c(187) | objective-c(184) |
| mysql(116) | templates(62) | iphone(47) |
| html(86) | stl(53) | cocoa-touch(39) |
| css(57) | linux(48) | osx(35) |
| regex(40) | subjective(45) | mac(34) |
| jquery(37) | pointers(44) | iphone-sdk(20) |
| sql(27) | java(42) | xcode(18) |
| ajax(26) | bash(40) | cocoa-bindings(18) |
| apache(23) | boost(31) | core-graphics(18) |

### C. Comparison with other Methods

We now want to evaluate whether a user is correctly assigned to the right interest group, and to which extent the user belongs to the interest group. To achieve this, we invited volunteers to manually label 902 users (co_answer_10 dataset) as the ground-truth and assigned each user with three group labels, chosen from 8 pre-defined labels: 'c-development', 'java-development', 'c#-development', 'web-development', 'ios-development', 'database', 'linux-development' and 'other-topic'. For example, user A sequentially has three labels: 'java-development','web-development','ios-development'. It means that user A has a big interest in the 'java-development' group, a medium interest in the 'web-development' group, a lower interest in the 'ios-development' group. We asked another volunteer (who was not involved in labeling the ground-truth) to label the results of the methods with the same 8 labels. As SLPA algorithm can detect overlapping communities. She was asked to assign an interest group name, from the 8 labels, to each community according to user tag lists in each community, then each user gets at least one interest group name. Besides, SLPA algorithm can evaluate to which extent a user belongs to a community by the frequency (a 'Post-process' in SLPA algorithm). Combined with the interest group name we assigned for each community, SLPA algorithm now can output an ordered interest group name list for each user. A clustering algorithm can only generate one cluster id for each user, so she was asked to assign an interest group name, from the 8 labels, for each cluster. The LDA method can give the probability of membership to

[3]Scikit-learn toolkit:
http://scikit-learn.org/stable/modules/clustering.html#spectral-clustering

each topic. The highest probability indicates that the user is more interested in that group. She only needed to associate the detected 30 topics with 8 group labels. Then we can get an ordered interest group name list for each user, sorted by probability. Like the LDA method, our approach can also give the probability membership to each topic, so she only needed to associate the detected 30 topics with 8 group labels. Then we can get an ordered interest group name list for each user, sorted by probability. Here, we only choose the top 3 group names for each user. Since each user has an ordered label list, we have to evaluate both the correctness of detected groups and the correctness of the order, therefore we use Normalized DCG (NDCG) to evaluate this. In our scenario, a NDCG@p value of 1.0 means detected interests and their order are totally the same as the ground-truth till position p, while a value between 0.0 to 1.0 means that the result are partially correct or ordered incorrectly. Fig 1 show the result of NDCG performance for each method.

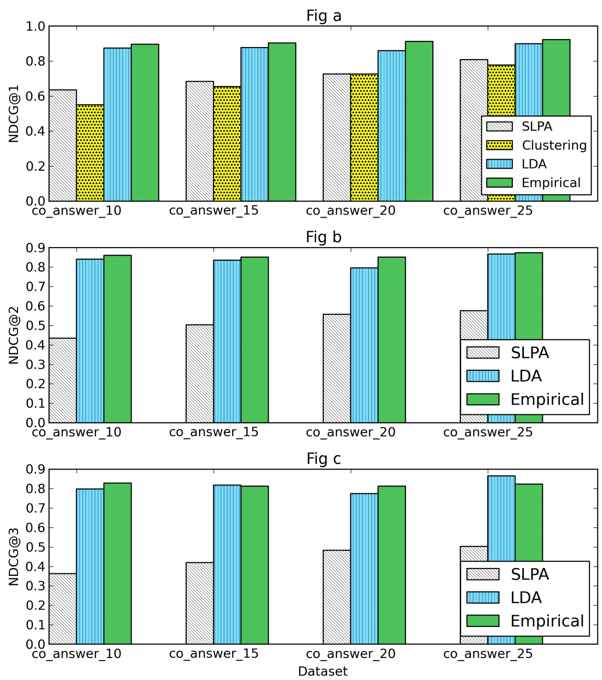


Fig. 1: NDCG results comparaison

NDCG@1 reflects the prominent interest detected by each algorithm compared with the ground-truth of prominent interest. We noticed that our empirical method is partially better than LDA, and outperforms SLPA and hierarchical clustering. We also mention that with the dataset becoming less noisy (people have prominent and clear-intention interests), all performances increase. The same phenomenon is also observed in NDCG@2,3. As hierarchical clustering algorithms give a hard partition there are no performance comparison for hierarchical clustering algorithm in NDCG@2,3.

## V. Conclusion

In this paper, we addressed the problem of detecting overlapping interest groups in Q&A sites. By empirically studying a dataset from the popular Q&A StackOverflow site, we proposed a question tag enrichment method and a tag prefix tree based topic extraction model. We then used this information to detect overlapping interest groups. We conducted experiments on a StackOverflow dataset with different approaches to provide a comparison and samples of results were analyzed in-depth. Results indicate that for this type of web communities our method is much simpler and faster, and that it can be a good replacement for other complicated methods in detecting overlapping interest groups. There are also limitations of our work and in particular our model requires each question to have several tags to indicate the domain of the question. In addition, we also need to point out that the involved human judgments may lead to some bias. There are many potential future directions for this work. An interesting one is to track the evolution of interest group and the evolution of user interests.

## Acknowledgment


The authors would like to thank StackOverflow for sharing their data. We also sincerely thank volunteers for helping us label the dataset for evaluation. We also appreciated very helpful advices from anonymous reviewers.

This work is supported by the ANR Ocktopus project.


## References


[1] F. Riahi, Z. Zolaktaf, M. Shafiei, and E. Milios, "Finding expert users in community question answering," in *Proceedings of the 21st international conference companion on World Wide Web*. ACM, 2012, pp. 791–798.

[2] J. Sung, J.-G. Lee, and U. Lee, "Booming up the long tails: Discovering potentially contributive users in community-based question answering services." in *ICWSM*, E. Kiciman, N. B. Ellison, B. Hogan, P. Resnick, and I. Soboroff, Eds. The AAAI Press, 2013. [Online]. Available: http://dblp.uni-trier.de/db/conf/icwsm/icwsm2013.html#SungLL13

[3] B. Li and I. King, "Routing questions to appropriate answerers in community question answering services," in *Proceedings of the 19th ACM international conference on Information and knowledge management*. ACM, 2010, pp. 1585–1588.

[4] J. Xie, S. Kelley, and B. K. Szymanski, "Overlapping community detection in networks: The state-of-the-art and comparative study," *ACM Comput. Surv.*, vol. 45, no. 4, p. 43, 2013.

[5] Y.-Y. Ahn, J. P. Bagrow, and S. Lehmann, "Link communities reveal multiscale complexity in networks," *Nature*, vol. 466, no. 7307, pp. 761–764, 2010.

[6] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney, "Statistical properties of community structure in large social and information networks," in *Proceedings of the 17th international conference on World Wide Web*. ACM, 2008, pp. 695–704.

[7] Z. Xu, Y. Ke, Y. Wang, H. Cheng, and J. Cheng, "A model-based approach to attributed graph clustering," in *SIGMOD Conference*, 2012, pp. 505–516.

[8] U. Gargi, W. Lu, V. S. Mirrokni, and S. Yoon, "Large-scale community detection on youtube for topic discovery and exploration," in *ICWSM*, 2011.

[9] X. Sun and H. Lin, "Topical community detection from mining user tagging behavior and interest." *JASIST*, vol. 64, no. 2, pp. 321–333, 2013.

[10] J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, and Z. Su, "Arnetminer: extraction and mining of academic social networks," in *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2008, pp. 990–998.

[11] A. Y. Ng, M. I. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," in *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS*. MIT Press, 2001, pp. 849–856.

[12] S. Chang and A. Pal, "Routing questions for collaborative answering in community question answering," in *Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, ser. ASONAM '13. New York, NY, USA: ACM, 2013, pp. 494–501.